Concepts of OOP

- Introduction to OOP
- Procedural Vs. Object Oriented Programming
- Principles of OOP
- Benefits and applications of OOP

Introduction to OOP

- OOP is a design philosophy. stands for Object Oriented It Programming.
- C++ was founded in (1983)

Bjarne Stroustrup

Introduction to OOP

- Object-**O**riented **P**rogramming (*OOP*) uses a different set of programming languages than old procedural programming languages like (*C, Pascal*, etc.).
- Everything in *OOP* is grouped as self sustainable "*objects*".

What is Object?

Pen

Board

Laptop

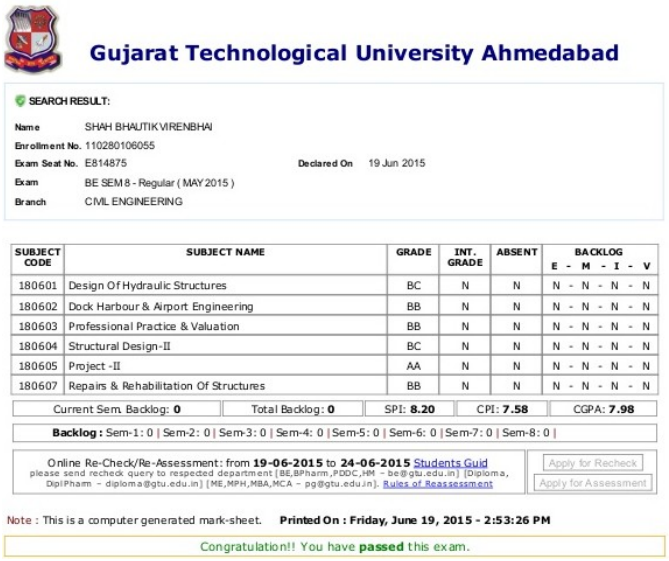
Bench

Student

Projector

Physical objects…

What is Object?



Gujarat Technological University Ahmedabad

SEARCH RESULT:

Name SHAH BHAUTIK VIRENBHAI
Enrollment No. 110280106055
Exam Seat No. E814875 Declared On 19 Jun 2015
Exam BE SEM 8 - Regular (MAY 2015)
Branch CIVIL ENGINEERING

SUBJECT CODE	SUBJECT NAME	GRADE	INT. GRADE	ABSENT	BACKLOG E - M - I - V
180601	Design Of Hydraulic Structures	BC	N	N	N - N - N - N
180602	Dock Harbour & Airport Engineering	BB	N	N	N - N - N - N
180603	Professional Practice & Valuation	BB	N	N	N - N - N - N
180604	Structural Design-II	BC	N	N	N - N - N - N
180605	Project -II	AA	N	N	N - N - N - N
180607	Repairs & Rehabilitation Of Structures	BB	N	N	N - N - N - N

Current Sem. Backlog: **0** | Total Backlog: **0** | SPI: **8.20** | CPI: **7.58** | CGPA: **7.98**

Backlog : Sem-1: 0 | Sem-2: 0 | Sem-3: 0 | Sem-4: 0 | Sem-5: 0 | Sem-6: 0 | Sem-7: 0 | Sem-8: 0 |

Online Re-Check/Re-Assessment: from **19-06-2015** to **24-06-2015** Students Guid
please send recheck query to respected department [BE,BPharm,PDDC,HM – be@gtu.edu.in] [Diploma, DipIPharm – diploma@gtu.edu.in] [ME,MPH,MBA,MCA – pg@gtu.edu.in]. Rules of Reassessment

Apply for Recheck | Apply for Assessment

Note : This is a computer generated mark-sheet. **Printed On : Friday, June 19, 2015 - 2:53:26 PM**

Congratulation!! You have **passed** this exam.

Result

Account

Bank Account

Logical objects…

Attributes and op

Attributes:
Name
Age
Weigh
t

Operations:
Eat
Sleep
Walk

Attributes
:
Company
Model
Weight

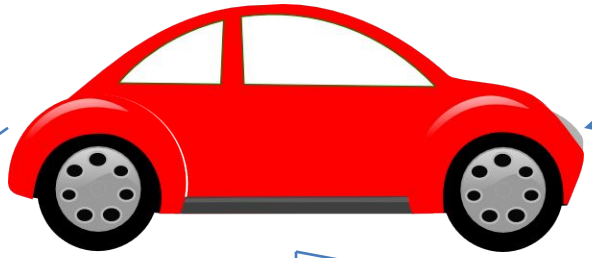
Operations:
Drive
Stop
FillFuel

Attributes:
AccountNo
HolderName
Balance

Operations:
Deposit
Withdraw
Transfer

Write down 5 objects with its attributes and operations

What is Object ?



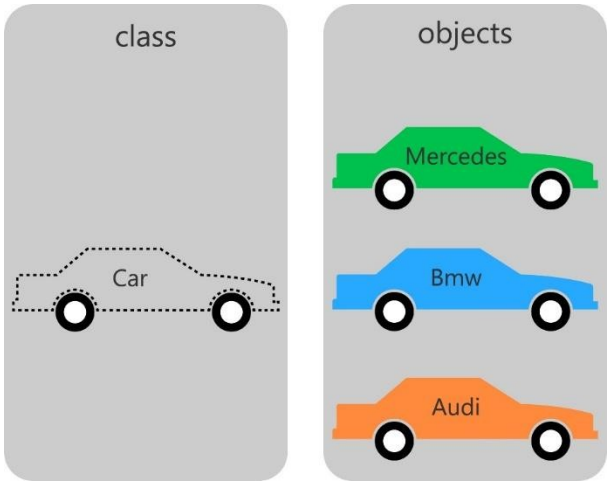
OBJECT:
CAR

Properties (Describe)
Manufacturer
Model
Color
Year
Price

Methods (Actions)
Start
Drive
Park

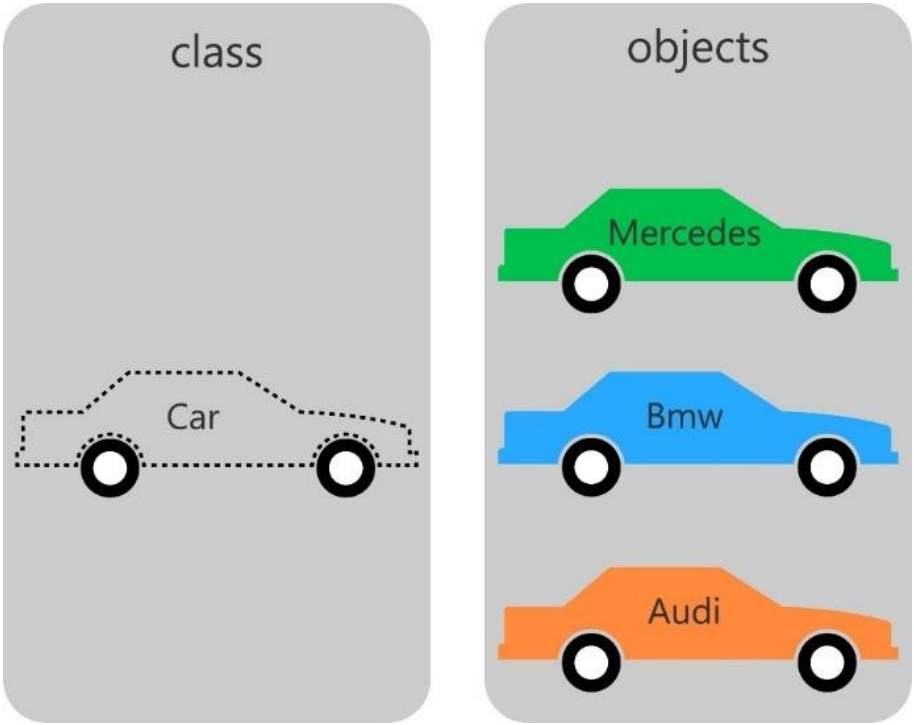
Events
On_Start
On_Parked
On_Brake

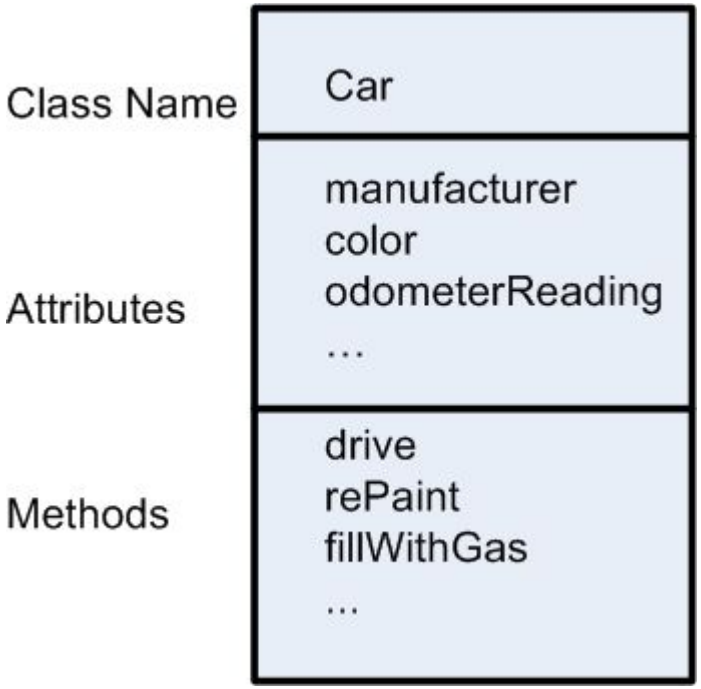
Classes…



Class: Blueprint (template) for object.
Object: Instance of class.

Class


class
Car
objects
Mercedes
Bmw
Audi


Class Name
Car
Attributes
manufacturer
color
odometerReading
…
Methods
drive
rePaint
fillWithGas
…

Applications of OOP

- Real Time Systems Design
- Simulation and Modeling System
- Object Oriented Database
- Client-Server System
- Neural Networking and Parallel Programming
- Decision Support and Office Automation Systems
- CIM/CAD/CAM Systems
- AI and Expert Systems

Procedural Vs. Object Oriented Programming

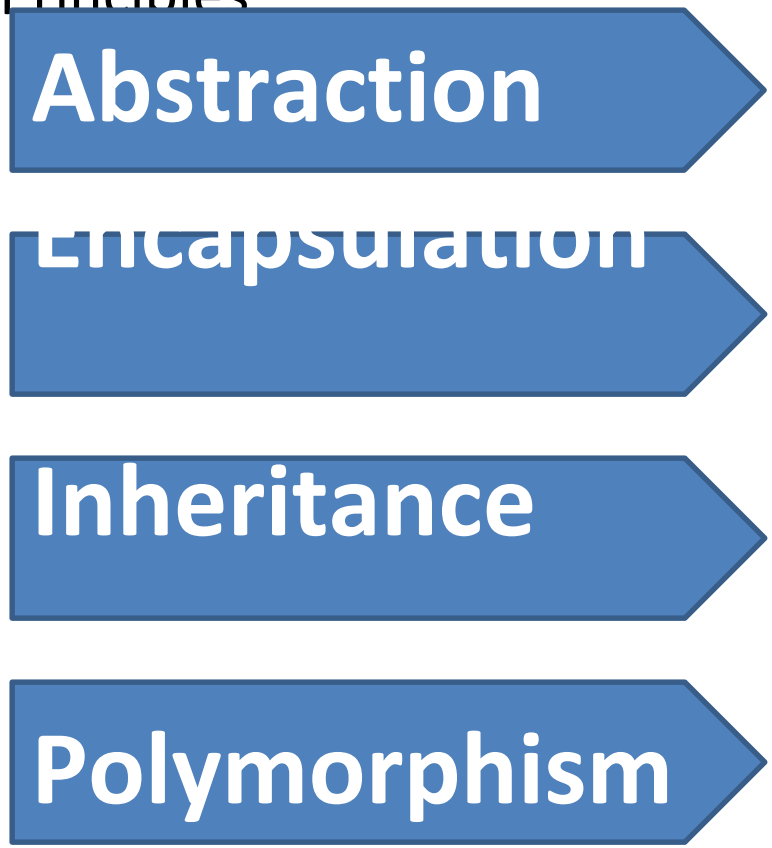
POP	OOP
Emphasis is on doing things not on data, means it is function driven	Emphasis is on data rather than procedure, means object
Main focus is on the function and	Main focus is on the data that is
Top Down approach in program	Bottom Up approach in program
Large programs are divided into smaller programs known as functions	Large programs are divided into classes and objects
Most of the functions share global data	Data is tied together with function in the data structure

Procedural Vs. Object Oriented Programming

POP	OOP
Data moves openly in the system from one function to another function	Data is hidden and cannot be accessed by external functions
Adding of data and function is difficult	Adding of data and function is easy
Wecannot declare namespace directly	We can use name space directly, Ex: using namespace std;
Concepts like inheritance, polymorphism, data encapsulation, abstraction, access specifiers are not available.	Concepts like inheritance, polymorphism, data encapsulation, abstraction, access specifiers are available and can be used easily
Examples: C, Fortran, Pascal, etc…	Examples: C++, Java, C#, etc…

Principles of OOP (A.E.I.P)

- There are mainly four OOP Principles

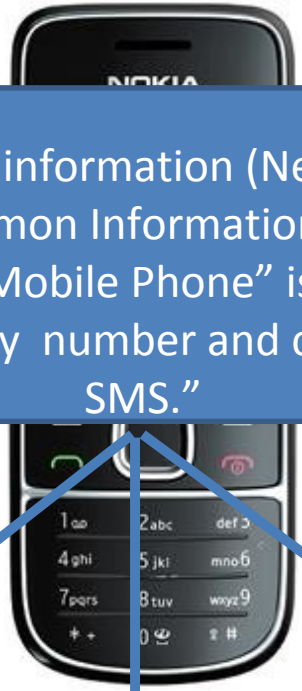


Abstractio

- **Abstraction** refers to the act of representing essential features without including the background details or explanations.
- **Abstraction** provides you a generalized view of your classes or object by providing relevant information.
- **Abstraction** is the process of hiding the working style of an object, and showing the information of an object in understandable manner.

Abstraction

E›



Abstract information (Necessary and Common Information) for the object "Mobile Phone" is make a call to any number and can send SMS."

Nokia 1400	Nokia 2700	Nokia 1400
Features: Calling SMS	Features: Calling SMS FM Radio MP3 Camera	Features: Calling SMS FM Radio MP3 Camera Video Recording Reading E-mails

Abstraction Example

Example:

If somebody in your collage tell you to fill application form, you will fill your details like **name, address, data of birth, which semester, percentage you have got** etc.

- If some doctor gives you an application to fill the details, you will fill the details like **name, address, date of birth, blood group, height and weight**.
- See in the above example what is the common thing? **Age, name, address** so you can create the class which consist of common thing that is called abstract class.

  That class is not complete and it can inherit by other class.

Encapsulation

- The wrapping up of data and functions into a single unit is known as **encapsulation**
- The insulation of the data from direct access by the program is called **data hiding** or **information hiding**.
- It is the process of enclosing one or more details from outside world through access right.

Encapsulation

- **Encapsulation** is the process of combining data and functions into a single unit called class. In Encapsulation, the data is not accessed directly; it is accessed through the functions present inside the class.
- Users are unaware about working of circuitry and hardware devices.

- **Abstraction** is a process where you show only "relevant" data and "hide" unnecessary details of an object from the user.
- Consider your mobile phone, you just need to know what buttons are to be pressed to send a message or make a call, What happens when you press a button, how your messages are sent, how your calls are connected is all abstracted away from the user.

Abstraction Vs Encapsulation

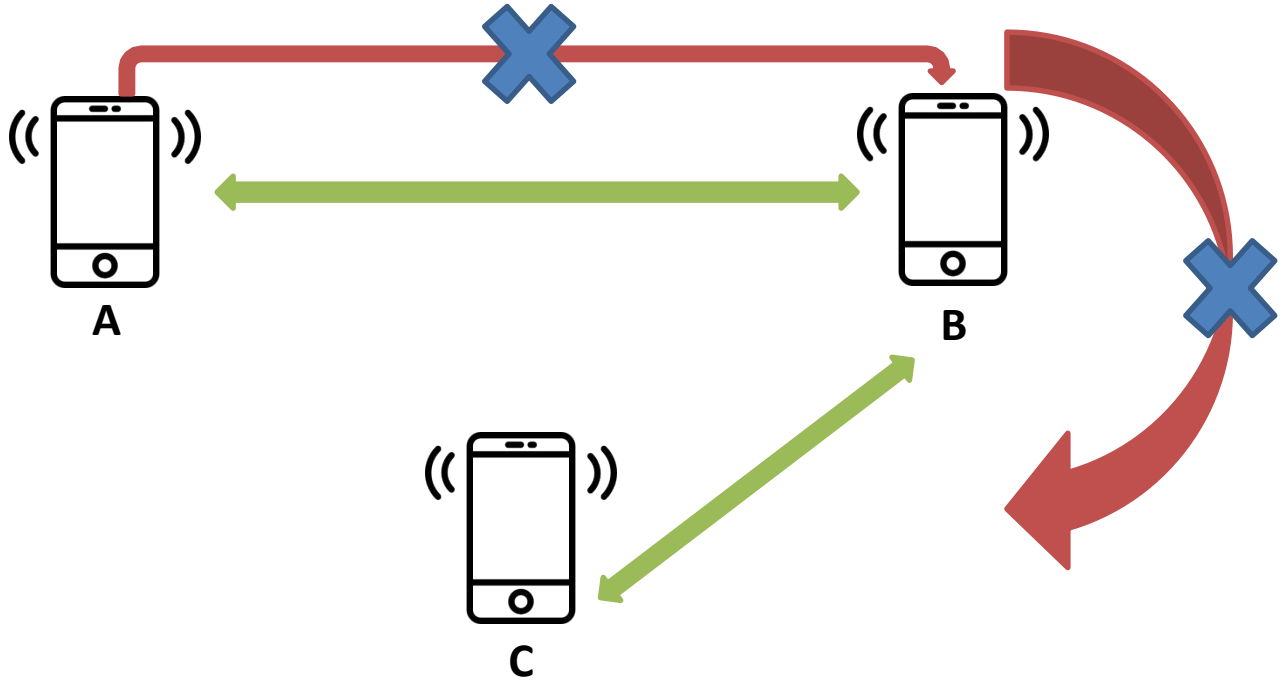
- Abstraction says what details to be made visible & Encapsulation provides the level of access right to that visible details.

Example:

- When we switch on the Bluetooth I am able to connect another mobile but not able to access the other mobile features like dialling a number, accessing inbox etc. This is because, Bluetooth feature is given some level of **abstraction**.

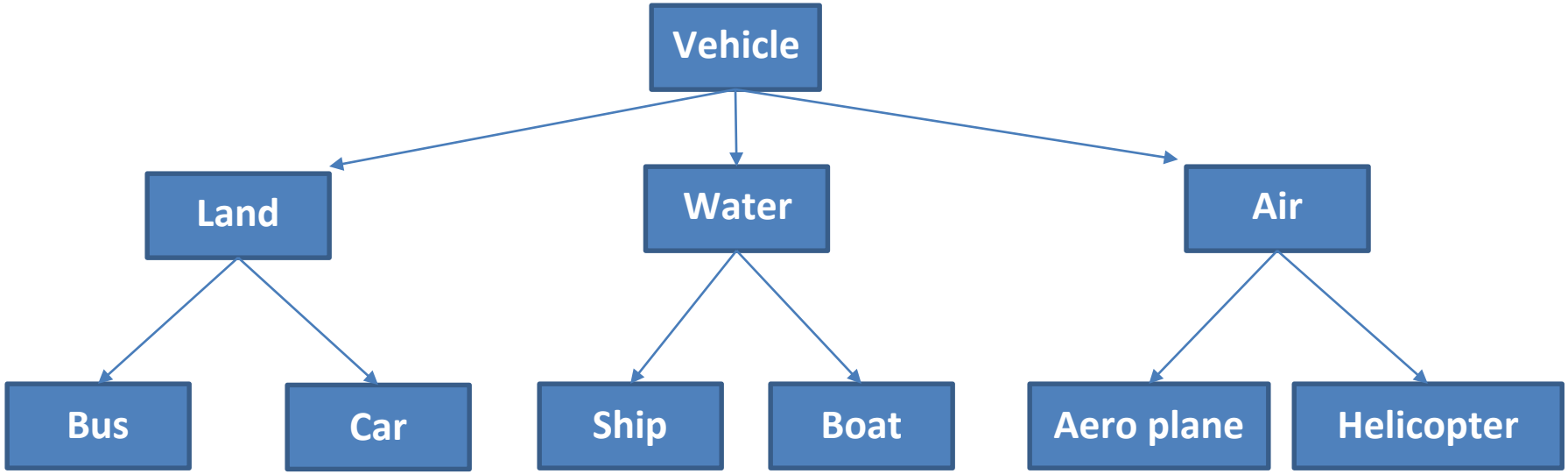
Abstraction Vs Encapsulation

When mobile A is connected with mobile B via Bluetooth whereas mobile B is already connected to mobile C then A is not allowed to connect C via B. This is because of accessibility restriction.



Inheritance

- **Inheritance** is the process by which objects of one class acquire the properties of objects of another class.



- Here Vehicle class can have properties like Chassis no. , Engine, Colour etc.
- All these properties inherited by sub classes of vehicle class.

Polymorphism

- **Polymorphism** means ability to take more than one form.
- For example the operation **addition**.
- For two numbers the operation will generate a **sum**.
- If the operands are strings, then the operation would produce a third string by **concatenation**.

C++ Object

in C++, Object is a real world entity, for example, chair, car, pen, mobile, laptop etc.

In other words, object is an entity that has state and behavior. Here, state means data and behavior means functionality.

Object is a runtime entity, it is created at runtime.

Object is an instance of a class. All the members of the class can be accessed through object.

Let's see an example to create object of student class using s1 as the reference variable

.

```
Student s1;  //creating an object of Student
```

C++ Class

In C++, class is a group of similar objects. It is a template from which objects are created. It can have fields, methods, constructors etc.

Let's see an example of C++ class that has three fields only.

```
class Student
 {
    public:
    int id;  //field or data member
    float salary; //field or data member
    String name;//field or data member
 }
```

C++ Class

In C++, class is a group of similar objects. It is a template from which objects are created. It can have fields, methods, constructors etc.

Let's see an example of C++ class that has three fields only.

```
class MyClass
 {                                  // The class
  public:                           // Access specifier
    int myNum;                     // Attribute (int variable)
    string myString;              // Attribute (string variable)
};
```

C++ Object and Class Example

```
#include <iostream>
using namespace std;
class Student {
  public:
    int id;//data member (also instance variable)
    string name;//data member(also instance variable)
};
int main() {
   Student s1; //creating an object of Student
   s1.id = 201;
   s1.name = "Sonoo Jaiswal";
   cout<<s1.id<<endl;
   cout<<s1.name<<endl;
   return 0;
}
```

C++ Object and Class Example

```
#include <iostream>
using namespace std;
class Student {
  public:
    int id;//data member (also instance variable)
    string name;//data member(also instance variable)
};
int main() {
   Student s1; //creating an object of Student
   s1.id = 201;
   s1.name = "Sonoo Jaiswal";
   cout<<s1.id<<endl;
   cout<<s1.name<<endl;
   return 0;
}
```

C++ Class Example: Initialize and Display data through method

```
#include <iostream>
using namespace std;
class Student {
  public:
     int id;//data member (also instance variable)
     string name;//data member(also instance variable)
     void insert(int i, string n)
     {
        id = i;
        name = n;    }
     void display()
     {
        cout<<id<<"  "<<name<<endl;
     }   };
int main(void) {
   Student s1; //creating an object of Student
   Student s2; //creating an object of Student
   s1.insert(201, "Sonoo");
   s2.insert(202, "Nakul");
   s1.display();
   s2.display();
   return 0;
}
```

C++ Class Example: Store and Display Employee Information

```
#include <iostream>
using namespace std;
class Employee {
  public:
    int id;//data member (also instance variable)
    string name;//data member(also instance variable)
    float salary;
    void insert(int i, string n, float s)
     {
        id = i;
        name = n;
        salary = s;
     }
    void display()
     {
        cout<<id<<"  "<<name<<"  "<<salary<<endl;
     }   };
```

C++ Class Example: Store and Display Employee Information

```
int main(void) {
    Employee e1; //creating an object of Employee
    Employee e2; //creating an object of Employee
    e1.insert(201, "Sahil",990000);
    e2.insert(202, "Nakul", 29000);
    e1.display();
    e2.display();
    return 0;
}
```

C++ Constructor

A constructor in C++ is a **special method** that is automatically called when an object of a class is created.

To create a constructor, use the same name as the class, followed by parentheses ():

There can be two types of constructors in C++.

Default constructor
Parameterized constructor

C++ Constructor

C++ Default Constructor

A constructor which has no argument is known as default constructor. It is invoked at the time of creating object. Let's see the simple example of C++ default Constructor.

C++ Constructor

C++ Default Constructor

```
#include <iostream>
using namespace std;
class Employee
 {
   public:
        Employee()
        {
            cout<<"Default Constructor Invoked"<<endl;
        }
};
int main(void)
{
    Employee e1; //creating an object of Employee
    Employee e2;
    return 0;
}
```

C++ Parameterized constructor

```
#include<iostream>
#include<conio.h>
using namespace std;
class Example {
     // Variable Declaration
     int a, b;
public:
     //Constructor

     Example(int x, int y) {
          // Assign Values In Constructor
          a = x;
          b = y;
          cout << "Im Constructor\n";
     }
     void Display() {
          cout << "Values :" << a << "\t" << b;
     }
};
int main() {
     Example Object(10, 20);
     // Constructor invoked.
     Object.Display();
     // Wait For Output Screen
     getch();
     return 0;
}
```

C++ Copy Constructor

The copy constructor in C++ is used to copy data of one object to another.

```
#include <iostream>
using namespace std;

// declare a class
class Wall {
  private:
    double length;
    double height;

  public:

Wall(double len, double hgt) {
      length = len;
      height = hgt;
    }

// copy constructor with a Wall object as
parameter
    // copies data of the obj parameter
    Wall(Wall &obj) {
      length = obj.length;
      height = obj.height;
    }

    double calculateArea() {
      return length * height;
    }
};
```

C++ Copy Constructor

The copy constructor in C++ is used to copy data of one object to another.

```
int main() {

  Wall wall1(10.5, 8.6);

  Wall wall2 = wall1;

cout << "Area of Wall 1: " << wall1.calculateArea() << endl;
  cout << "Area of Wall 2: " << wall2.calculateArea();

  return 0;
}
```

C++ Destructor

A destructor works opposite to constructor; it destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically.

Note: C++ destructor cannot have parameters. Moreover, modifiers can't be applied on destructors.

A destructor is defined like constructor. It must have same name as class. But it is prefixed with a tilde sign (~).

C++ Constructor and Destructor Example

```
#include <iostream>
using namespace std;
class Employee
 {
   public:
       Employee()
       {
          cout<<"Constructor Invoked"<<endl;
       }
       ~Employee()
       {
          cout<<"Destructor Invoked"<<endl;
       }
};
int main(void)
{
    Employee e1; //creating an object of Employee
    Employee e2; //creating an object of Employee
    return 0;
}
```

Constructor Overloading in C++

Overloading in Constructors are the constructors with the same name and different parameters (or arguments). Hence, the constructor call depends upon data types and the number of arguments.

Constructor Overloading in C++

```
#include <iostream>
using namespace std;

class constructor
{

public:
	float area;

	// Constructor with no parameters
	constructor()
	{
		area = 0; //Giving value 0 to variable area
	}

	// Constructor with two parameters a and b
	constructor(int a, int b)
	{
		area = a * b;
	}

	void display() //member function of the class
	{
		cout<< "Area: "<<area<< endl;
	}
};

int main()
{
	constructor obj;
	constructor obj2( 22, 40);

	obj.display();
	obj2.display();
	return 1;
}
```

Static Keyword in C++

Static keyword has different meanings when used with different types. We can use static keyword with:

Static Variables : Variables in a function, Variables in a class
Static Members of Class : Class objects and Functions in a class

Static Variables

Static variables in a Function: When a variable is declared as static, space for **it gets allocated for the lifetime of the program**.

```
#include <iostream>
#include <string>
using namespace std;
void demo()
{
	static int count = 0;
	cout << count << " ";

	// value is updated and
	// will be carried to next
	// function calls
	count++;
}
int main()
{
	for (int i=0; i<5; i++)
		demo();
	return 0;
}
```

Static Member functions

The static member functions are special functions used to access the static data members or other static member functions.

A member function is defined using the static keyword. A static member function shares the single copy of the member function to any number of the class' objects.

We can access the static member function using the class name or class' objects. If the static member function accesses any non-static data member or non-static member function, it throws an error.

Syntax

class_name::function_name (parameter);

Example

```
#include <iostream>
using namespace std;
class Note
{
// declare a static data member
static int num;

public:
// create static member function
static int func ()
{
return num;
}
};
// initialize the static data member using the class name and the scope resolution operator
int Note :: num = 5;
int main ()
{
// access static member function using the class name and the scope resolution
cout << " The value of the num is: " << Note:: func () << endl;
return 0;
}
```

Example- Let's create another program to access the static member function using the class' object in the C++ programming language.

```
#include <iostream>
using namespace std;
class Note
{
static int num;

public:
// create static member function
static int func ()
{
cout << " The value of the num is: " << num << endl;
}
};
```

Example- Let's create another program to access the static member function using the class' object in the C++ programming language.

```
int Note :: num = 15;

int main ()
{
   // create an object of the class Note
   Note n;
// access static member function using the object
n.func();

return 0;
}
```

C++ Strings

In C++, string is an object of **std::string** class that represents sequence of characters. We can perform many operations on strings such as concatenation, comparison, conversion etc

C++ String Example

```
#include <iostream>
using namespace std;
int main( ) {
    string s1 = "Hello";
        char ch[] = { 'C', '+', '+'};
        string s2 = string(ch);
        cout<<s1<<endl;
        cout<<s2<<endl;
}
```

C++ String copy Example

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
   char key[25], buffer[25];
   cout << "Enter the key string: ";
   cin.getline(key, 25);
   strcpy(buffer, key);
   cout << "Key = "<< key << endl;
   cout << "Buffer = "<< buffer<<endl;
   return 0;
}
```

C++ String length Example

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char ary[] = "Welcome to C++ Programming";
    cout << "Length of String = " << strlen(ary)<<endl;
    return 0;
}
```

Function	Description
int compare(const string& str)	It is used to compare two string objects.
int length()	It is used to find the length of the string.
void swap(string& str)	It is used to swap the values of two string objects.
string substr(int pos,int n)	It creates a new string object of n characters.
int size()	It returns the length of the string in terms of bytes.
void resize(int n)	It is used to resize the length of the string up to n characters.
string& replace(int pos,int len,string& str)	It replaces portion of the string that begins at character position pos and spans len characters.
string& append(const string& str)	It adds new characters at the end of another string object.
char& at(int pos)	It is used to access an individual character at specified position pos.

BASIS FOR COMPARISON	CHARACTER ARRAY	STRING
Basic	Character array is collection of variables, of character data type.	String is class and variables of string are the object of class "string".
Syntax	char array_name [size];	string string_name;
Indexing	An individual character in a character array can be accessed by its index in array.	In string the particular character can be accessed by the function "string_name.charAt(index)".
Data Type	A character array does not define a datatype.	A string defines a datatype in C++.
Operators	Operators in C++ can not be applied on character array.	You may apply standard C++ operator on the string.
Boundary	Array boundaries are	Boundaries will not overrun.

Operations on Strings

1) Input Functions
2) Capacity Functions
3) Iterator Functions
4) Manipulating Functions:

C++ Operator Overloading

C++ operator overloading is one of the most powerful features of C++ that allows a user to change the way the operator works.

In C++, we can change the way operators work for user-defined types like objects and structures. This is known as **operator overloading**.

C++ has the ability to prove the operators with a special meaning for a data type. So the mechanism of adding special meaning to an operator is called operator overloading.

Why do we need Operator overloading in C++?

Operators in C++ like +, -, *, / can operate in datatypes like int, float, double etc as predefined operational meanings. But these operators can't operate in user-defined datatypes like objects without extension or adding some sort of code to alter their operational meaning.

Such a way of extending the operational functionality of certain operators in C++ is called **operator overloading**.

C++ operator overloading : Syntax

```
return_type  operator  operator_symbol
(argument_list)
{

 //body of function
}
```

Operator overloading in C++ can be achieved in following ways

- Operator overloading using member function
- Operator overloading using non-member function
- Operator overloading using friend function

which we cannot?

- pointer to member access operator (.*)
- scope resolution operator (::)
- member access operator (.)
- condition operator (?:)
- size operator (sizeof)
- run-time type information operator (typeid)

Unary Operator Overloading

As the name suggests, Unary operators operate on single operand or data.

Following are the examples of Unary operators:

- Decrement (—) and Increment (++) operator

Note : If increment/decrement operators are used before variable, they are called prefix operators i.e ++x. And if increment/decrement operators are used after variable, they are called postfix operators i.e x++.

Unary Operator Overloading

```
#include <iostream>
using namespace std;
class Test
{
   private:
      int num;
   public:
       Test(): num(8){}
       void operator ++()        {
          num = num+2;
       }
       void Print() {
           cout<<"The Count is: "<<num;
       }
};
int main()
{
    Test tt;
    ++tt;  // calling of a function "void operator ++()"
    tt.Print();
    return 0;
}
```

Unary Operator Overloading

```
// Overload ++ when used as prefix
#include <iostream>
using namespace std;

class Count {
  private:
   int value;

  public:
  Count() : value(5) {}

   // Overload ++ when used as prefix
   void operator ++ () {
      ++value;
   }
   void display() {
      cout << "Count: " << value <<
endl;
   }};
```

```
int main() {
   Count count1;

   // Call the "void operator
++ ()" function
   ++count1;

   count1.display();
   return 0;
}
```

Binary Operator Overloading

As the name suggests, those operators which operate on two operands or data are called binary operators.

Here is an example to show how binary operators are overloaded in C++.

Binary Operator Overloading

```
// C++ program to overload the binary operator +
// This program adds two complex numbers

#include <iostream>
using namespace std;

class Complex {
   private:
    float real;
    float imag;

   public:
    // Constructor to initialize real and imag to 0
    Complex() : real(0), imag(0) {}

    void input() {
        cout << "Enter real and imaginary parts respectively: ";
        cin >> real;
        cin >> imag;
    }

    // Overload the + operator
    Complex operator + (const Complex& obj) {
        Complex temp;
         temp.real = real + obj.real;
         temp.imag = imag + obj.imag;
         return temp;
    }
```

Binary Operator Overloading

```
    void output() {
        if (imag < 0)
            cout << "Output Complex number: " << real << imag << "i";
        else
            cout << "Output Complex number: " << real << "+" << imag << "i";
    }
};

int main() {
    Complex complex1, complex2, result;

    cout << "Enter first complex number:\n";
    complex1.input();

    cout << "Enter second complex number:\n";
    complex2.input();

   // complex1 calls the operator function
   // complex2 is passed as an argument to the function
    result = complex1 + complex2;
    result.output();

    return 0;
}
```

Output

```
}
Class Myclass
{
    private:
    int member1;
}
int main()
{
    Myclass obj;
    obj.member1=5;
}
```

C++ Friend function

If a function is defined as a friend function in C++, then the protected and private data of a class can be accessed using the function.

By using the keyword friend compiler knows the given function is a friend function.

For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword friend.

C++ Friend function

Declaration of friend function in C++.

```
class class_name
{
    friend data_type function_name(argument/s);
// syntax of
friend function.
};
```

In the above declaration, the friend function is preceded by the keyword friend. The function can be defined anywhere in the program like a normal C++ function. The function definition does not use either the keyword **friend or scope resolution operator**.

C++ function Example

```
#include <iostream>
using namespace std;
class Box
{
    private:
        int length;
    public:
        Box(): length(0) { }

        friend int printLength(Box); //friend function
};
int printLength(Box b)
{
   b.length += 10;
    return b.length;
}
int main()
{
    Box b;
    cout<<"Length of box: "<< printLength(b)<<endl;
    return 0;
}
```

Let's see a simple example when the function is friendly to two classes.

```
#include <iostream>
using namespace std;
class B;          // forward declarartion.
class A
{
    int x;
    public:
    void setdata(int i)
    {
        x=i;
    }
    friend void min(A,B);         // friend function.
};
class B
{
    int y;
    public:
    void setdata(int i)
    {
        y=i;
}
```

Let's see a simple example when the function is friendly to two classes.

```
friend void min(A,B);                    // friend function  };

void min(A a,B b)
{
    if(a.x<=b.y)
    std::cout << a.x << std::endl;
    else
    std::cout << b.y << std::endl;
}
   int main()
{
   A a;
   B b;
   a.setdata(10);
   b.setdata(20);
   min(a,b);
    return 0;
 }
```

C++ Friend class

friend class can access both private and protected members of the class in which it has been declared as friend.

```
#include <iostream>
using namespace std;
class A
{
    int x =5;
    friend class B;          // friend class.
};
class B
{
  public:
    void display(A &a)
    {
        cout<<"value of x is : "<<a.x;
    }
};
```

```
int main()
{
    A a;
    B b;
    b.display(a);
    return 0;
}
```

Friend Function using Operator Overloading in C++

Introduction

- Friend function using operator overloading offers better flexibility to the class.
- These functions are not a members of the class and they do not have 'this' pointer.
- When you overload a unary operator you have to pass one argument.
- When you overload a binary operator you have to pass two arguments.
- Friend function can access private members of a class directly.

Syntax:

```
friend return-type operator operator-symbol (Variable 1, Varibale2)
{
    //Statements;
}
```

Program demonstrating Unary operator overloading using Friend function

```
#include<iostream>
using namespace std;
class UnaryFriend
{
    int a=10;
    int b=20;
    int c=30;
    public:
        void getvalues()
        {
            cout<<"Values of A, B & C\n";
            cout<<a<<"\n"<<b<<"\n"<<c<<"\n"<<endl;
        }
        void show()
        {
            cout<<a<<"\n"<<b<<"\n"<<c<<"\n"<<endl;
        }
        void friend operator-(UnaryFriend &x);      //Pass by reference
};
```

Program demonstrating Unary operator overloading using Friend function

```
int main()
{
    UnaryFriend x1;
    x1.getvalues();
    cout<<"Before Overloading\n";
    x1.show();
    cout<<"After Overloading \n";
    -x1;
     x1.show();
     return 0;
}
```

Output:

Values of A, B & C

https://mjginfologs.com/type-conversion-in-cpp/#:~:text=Class%20type%20to%20Basic%20Type,-The%20constructor%20functions&text=C%2B%2B%20allows%20us%20to%20define,)%20%7B%20%2F%2FProgram%20statement.%20%7D